**Introduction:** Types of languages – Procedural, Functional, Object Oriented

Object-> code + data

**Static vs Dynamic Languages:**

* **Static:** Type check at compile time
  + Errors will show at compile time
  + More control
  + Declare datatype before you use it
* **Dynamic:** Perform type check at runtime
  + Error might not show till program is run
  + No need to declare data type of variables
  + Might give error at compile time

**Introduction to JAVA:**

.java file -> **compile**-> .class file (byte code) -> **Interpreter** (line by line)->M/C code (0,1)

**Byte Code:**

We need JVM to run this code and converted to machine code.

Can run on all operating system.

**Architecture of Java:**

JDK = JRE + Development tools

JRE = JVM + Library Classes

JVM has JIT

**Static:**

Run function without creating object of the class.

**main:**

Entry point of java code

**String[] args:**

Command line arguments, we can pass these arguments from cmd

java Main “Devarshi”

**Change Bytecode Location:**

javac -d . Demo.java

javac -d .. Demo.java

**Package:**

Folder to locate java files

And to provide access

**Primitive Data Types in Java:**

Any data type which cannot be break into any other data type.

Int a = 10;-> 10 is literal , a is an identifier

**Type Casting:**

Float > integer

If asking for integer but giving float will not work but vice versa will work.

Int num = (int)(67.56f); 🡪 type casting 🡪 67

Ex: int a = 257;

byte b = (byte)(a);

b 🡪 1 (max value of byte is 256 so 1 is remainder of maximum value)

**Java follows UNICODE principle**

**Type promotion rules:**

Two types operation will give bigger data type return value

EX:

byte b = 42;

char c = ‘a’;

short s = 1024;

int I = 50000;

float f = 5.67f

double d = 0.1234;

double result = (f\*b) + (i/c) – (d-s);

* Float value 1777.016

**While loop:**

**For loop:**

**[Q1\_LargestNumber.java](https://github.com/Devarshi-tech/DSA_Java/blob/main/Codes/Q1_LargestNumber.java)**

**[Q2\_AlphabetCaseCheck](https://github.com/Devarshi-tech/DSA_Java/blob/main/Codes/Q2_AlphabetCaseCheck.java)**

**[Q3\_FibonacciNumbers](https://github.com/Devarshi-tech/DSA_Java/blob/main/Codes/Q3_FibonacciNumbers.java)**

[**Q4\_CountingOccurrences**](https://github.com/Devarshi-tech/DSA_Java/blob/main/Codes/Q4_CountingOccurrences.java)

[**Q5\_ReverseOfNumber**](https://github.com/Devarshi-tech/DSA_Java/blob/main/Codes/Q5_ReverseOfNumber.java)

Q6\_Calculator

**Switch Statements:**

switch(val){

case val1:

//do something

Break;

Case val2:

// do something

breake;

default:

//do something

}

* If break is not provided code will execute till the end

**Functions/Methods in Java:**

**Scope:**

Accessing variable locally or globally.

Block Scope: variable within a scope

Shadowing: changing global variable value in a block / it will hide global value of that variable

**Variable Arguments(Varargs):**

when to pass n number of arguments.

static void func(int …v){

// …v -> it will internal store in an array

System.out.println(Arrays.toString(v)); 🡪 [1,2,3 . . . etc]

}

static void func(String a, int …v){

// first pass string then pass n number of integers

// order is important

}

**Overloading:**

Two or more functions exists with same name and with different parameters.

[**Q7\_PrimeNumber**](https://github.com/Devarshi-tech/DSA_Java/blob/main/Codes/Q7_PrimeNumber.java)

[**Q8\_ArmStrongNumbers**](https://github.com/Devarshi-tech/DSA_Java/blob/main/Codes/Q8_ArmStrongNumbers.java)

**Arrays and Array List:**

Syntax: datatype [] variablName = new datatype[size];

datatype [] variableName = {val1, val2, val3};

collection of same data type.

int [] i = new int [5];

int [] i 🡪 reference variable/ declaration of array (i is getting defined in the stack) (compile time)

new int [5] 🡪 initialisation/ actually here object is being created in heap (run time/ dynamic memory allocation)

# Primitives are stored in stack memory

# Heap objects are not continuous (allocation of array data location in heap is not continuous)

# In java internally array objects may not be continuous -> depends on JVM

# position of array starts from zero

**String Array:**

String [] arr = new String [4];

arr [0] 🡪 null

arr-> store in stack memory

[\_, \_, \_, \_] -> 4 elements stored in heap and this each element itself is an object which has stored in different part of memory

All reference variables by default points to “null”

**Length:**

arr. length

**ForEach Loop:**

for (int num: arr) {

// for every element in array, num represents element of array

}

**Print Array:** Arrays.toString(arr);

**2D Arrays:**

Int [] [] arr2D = new int [3] [];

Length of column is not mandatory

Int [] [] arr2D = {

{},{},{}

}

**Dynamic Arrays:**

**Array List:** if we don’t know the size of array

(it is similar to vectors in C++)

ArrayList<String> list = new ArrayList<>(initialcapacity:10);

In peranthesis we can give initial length

<> -> in this we cannot pass primitive , can pass wrapper class only

list.add(“Hello”);

list.add(“world”);

System.out.println(list); 🡪 without toString()

**Internal working:** ArrayList<Integer> arr = new ArrayList<>();

arr 🡪 stored in stack

new ArrayList<>() 🡪 stored in heap

size is fixed internally but when array list fills by some amount then it will create a new arraylist that may be new size capacity and old elements copied to new list and old ones are deleted.

It has constant time complexity O(1).

ArrayList<ArrayList<String>> arr = new ArrayList<>();

**Q9\_SwappingValuesInArray**

**Q10\_MaxValueFromArray**

**Q11\_ReversingArrayValues**

**Linear Search Algorithm:**

**Q12\_FindWhether14ExistsInArray** – arr = [18,12,9,14,77,50]

Best case of linear search time complexity O(1)

Worst case O(N) 🡪 N is size of array

Best case going to be when searching for element is if element is at 0th place

How many checks will the loop make in best case i.e. element found at 0th index?

Linear time complexity is, when time is increasing linearly w.r.t size of the element.